HEAT MIDI Keyboard

Landyn Francis
Jacob Mealey

April 3, 2023

Abstract

The MIDI protocol is the primary specification for connecting electronic musical instruments. This report outlines the design and verification of a microprocessor−based device that will be used as an electronic piano that communicates via MIDI over USB. The device must meet many specifications. The device is powered using a USB cable and must regulate power with 75% efficiency or better. The device must have a full octave (twelve keys) of inputs that will be sampled and processed, then transferred via USB to a host device. These inputs must be velocity sensitive, that is to say, if the user presses the key faster the device reports it as such. It must be able to detect key presses in a range of 5cm/s to 20cm/s $\pm 10\%$. Along with these twelve inputs, the user must be able to input a desired volume and octave. The device meets all specifications.

Contents

List of Figures

List of Tables

1 Introduction

This report describes the design and operation of the HEAT MIDI Keyboard, a twelve−key Musical Instrument Device Interface (MIDI) keyboard solution. The keyboard allows the user to send MIDI data over a USB connection to a host device running a Digital Audio Workstation (DAW) program. HEAT is an acronym for Hall Effect Always Tracking. Hall Effect for the type of sensors used to detect key−presses, and Always Tracking due to the fact that the device is designed in a way where it is always tracking the position of each key. This report is organized as follows: The first section will introduce some background information, as well as the purpose and important operation notes for the keyboard. Following that, Section 2 will be a high level overview of the keyboard's functional blocks. Next, section 3 will provide further detail into the engineering design decisions made during the design process. Section 4 discusses the results of tests that were run on the keyboard to confirm it is functioning as designed.

1.1 Background

Digital music is the primary method of music distribution in the world. MIDI is a widely used standard for connecting and controlling electronic musical instruments. It was developed in 1982 as a standard meant to be universal and expandable, allowing an entire musical performance to be captured and edited digitally[4]. Any MIDI note message has two essential properties, the intended note, and the velocity. Like a traditional piano, a MIDI keyboard must be responsive to different velocities for a given note. If a key is pressed harder, it should create a louder, stronger sounding note, and vice versa. Vitally, a MIDI keyboard does not produce any of its own musical notes. Instead, the note information sent in a MIDI message to a MIDI synthesizer which produces the appropriate sound. For this project, a Digital Audio Workstation program on a PC is used as the synthesizer. Traditionally, these MIDI messages are sent over a proprietary MIDI cable. Instead, this device takes advantage of MIDI over USB [5]. This allows MIDI messages to be sent over the same port that will provide power to the device.

1.2 Purpose

A MIDI keyboard has two primary purposes, collecting the note and velocity data from a given key press and sending that data to a host device. The note can be determined from a standard piano style keyboard with an adjustable octave value. Many similar devices detect velocity with two buttons which are placed on opposite ends of the key. As the key travels, the first button is pressed, then the other, and the time between these presses can be used to determine the key velocity. By contrast, this project was designed using hall effect sensors and magnets. Magnets are placed on the bottom of each key, near the end. They are positioned above hall effect sensors on the board such that when the key is pressed, the magnet is moved closer to the sensor. Hall

effect sensors detect nearby magnetic field strength and output a corresponding analog voltage. This voltage can be used to determine the position, and then the velocity of a given key press. Optionally, MIDI can also send controller messages to the host device, to control other properties of a musical performance, such as volume, or pitch. This keyboard implements volume control through a volume dial on the board, as well as mute, through a switch built into the dial. This project was subject to a contract with a determined set of inputs, outputs, and specifications. The full contract can be seen in Appendix A. The first specification is the velocity sensitivity of the device. It must be able to detect velocities of $5 - 20cm/s \pm 10\%$. The next specification requires the device to meet standard MIDI protocol compliance. For this project, this is defined as being able to successfully send MIDI packets to a host device. The device must not make use of any development board. The device must be built on a custom−designed printed circuit board (PCB). The final specification requires the power regulation section of the device to be 75% efficient or better.

1.3 Operation

This section will cover the operation of the device. The device must be connected using the micro−USB port to a host device running a digital audio workstation (DAW). The device should be automatically detected as a MIDI device by the host device OS. Most DAW's require a simple setup for MIDI keyboards. This section will cover setup in FL Studio 20. First, the MIDI input must be enabled. Navigate to Options>System>MIDI Settings. The device will appear as Hall Effect MIDI Keyboard in the input section. Select the keyboard, then select Enable. Press a note on the keyboard to confirm the program is receiving the notes. The help bar in the top left of the program should display the played note and velocity. To assign an instrument, drag the desired instrument from the left Browser panel to the desired channel on the channel rack. Key presses should now result in a note being played by the desired instrument. The volume knob must also be configured in FL Studio. Using the Mixer, right−click the Master volume slider, and under remote control select link to controller. The dialog box requires the input of the port, channel, and controller. FL Studio has an auto−detect feature, so simply rotate the volume dial, and the software will assign the controller correctly. With the device setup, it can now be used to create music. Change the octave using the Octave Up or Octave Down buttons on the device. Rotate the volume dial to adjust the volume, and press the dial in to kill all active notes.

The design process will now be documented. Section 2 is a high level overview of the project, and its design.

2 Overview

This section will be a high−level discussion of the functionality of the project. The hardware and software sections will be discussed separately. The MIDI Keyboard must send MIDI messages, as well as display key press velocity.

2.1 Hardware

This section will discuss the necessary functions of the hardware for the MIDI Keyboard. The di−agram is comprised of nine blocks across three sections. The sections are input, signal processing and output. Figure 1 shows the hardware block diagram.

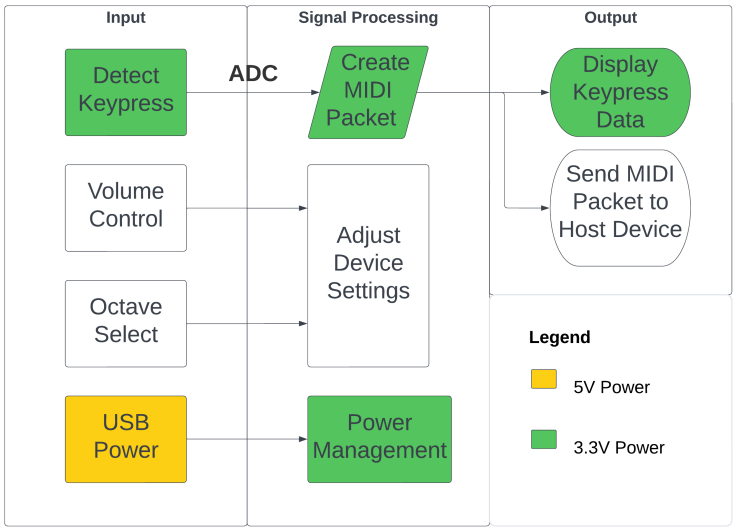

Fig. 1. The Hardware Block Diagram for the MIDI Keyboard.

2.1.1 Input

The primary input for the device is the key presses on the keyboard. This is done with a combina−tion of twelve hall effect sensors and magnets, and a twelve−channel analog−to−digital converter. The next two inputs are volume and octave control. These are both digital signals that can be sent directly to the signal processing section. The octave control is put into effect with two buttons to increment and decrement, while the volume is controlled with a rotary encoder. The last input is the power for the device to operate. This is done using a microUSB connection providing 5V USB power. This 5V is sent to the signal processing section to be converted for use in the device.

2.1.2 Signal Processing

The signal processing section is comprised of three main components. They are responsible for taking raw user input and making the device respond in the desired way. The first block is creating the MIDI message. This requires calculating the velocity and note for a given key press. Once these values are calculated the MIDI message is prepared and sent to the output section. The next block adjusts device settings and is done by processing incoming signals from the volume and octave control blocks and making the appropriate adjustments. The power management block is responsible for converting the 5V USB power to 3.3V power used by the device electronics. The Legend shows an overview of how power is distributed in the project.

2.1.3 Output

The output section contains the final blocks of the diagram and represents the outgoing signals from the device. The first block is sending MIDI message to the host device. This is done over a USB connection and represents the finish line for the primary function of the device. The other block is displaying key press data. As specified, the device must display the velocity of the most recent key press in cm/s on the LCD.

2.2 Software

This section will describe the software running on the MIDI Keyboard at a high level. Figure 2 is a block diagram showing an overview of the software functionality and flow.

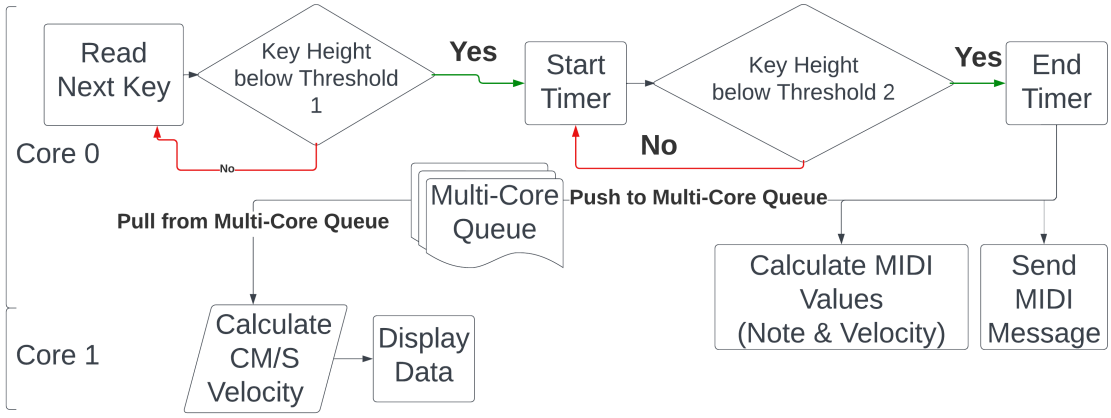

Fig. 2. The Software Block Diagram for the MIDI Keyboard

2.2.1 Multi−Core

The software is separated into Core 0 and Core 1 sections. This division of tasks was done so that Core 0 handles mission−critical operations and Core 1 handles non−critical code. Core 0 handles all communication with the ADC and the MIDI host. Core 1 is in charge of driving the LCD as well as calculating the velocity of the key press, to be displayed on the LCD. A thread−safe queue is used to move a copy of the keyboard state to Core 1, minimizing the shared memory. The division of these tasks means minimal coupling between the two cores, reducing the potential for either race conditions or time wasted waiting for locks.

2.2.2 Data Flow

Each key is read by sampling each channel of the ADC, one at a time until one of them is below a certain threshold. Two thresholds are used to detect a key press, one is halfway down the keys travel, and the other is near the bottom. The first threshold determines the start of a key press. On a traditional piano, the key must travel a distance before the hammer strikes the string to create sound. Placing the threshold halfway down the key travel replicates this behavior, and helps to avoid unintended key presses. Once this threshold is crossed, a timer is started. The second threshold determines whether the key has reached the bottom of its travel. Once this threshold is crossed, the aforementioned timer is stopped, and the data can be used to characterize the key press. The fall time is used as a velocity for the MIDI message and packaged with the note ready to be sent to the host device. The timing data is then sent to the other core. On Core 1, the velocity in centimeters per second is calculated and displayed on the LCD. In Section 3 design decisions for both the hardware and the software of the device will be explored.

3 Details

3.1 Hardware

The hardware for this project was split across two PCBs, which allowed the team to work in tandem and develop different sections at the same time. Using two fabrication lines was also a cost−saving measure. Developing separate PCBs allowed for the fabrication to be done using different methods, where one board is four layers, and requires very tight tolerances but is rel−atively small, and the other has two layers, and uses larger components but is relatively large. The two PCBs are called the Brainboard and the Keysboard. The Brainboard consists of the microcontroller, display, power management, octave select, and volume control. The Keysboard has twelve 3D printed keys, twelve hall effect sensors, a twelve−channel ADC, and the necessary reference circuit for ADC conversion.

3.1.1 Brainboard

The Brainboard is the primary PCB of the project. It contains the microcontroller, and all components for it to function properly, as well as the power supply, display, user input buttons, and volume dial. The primary purpose is to receive signals from the Keysboard, and package them into MIDI messages sent over USB to the host device. This section will detail the engineering decisions made in designing the Brainboard PCB and all circuits on it.

3.1.2 PCB Design

The Brainboard is a 4 layer board, with two power planes in the middle. This increases costs, but simplifies design, and improves electrical performance. Figure 3 shows the impedance controlled stackup provided by JLCPCB, that is used for the Brainboard.

a) JLC7628 Stackup:

Layer	Material Type	Thickness	
Top Layer1	Copper	0.035 mm	
Prepreg	7628*1	0.2104 mm	
Inner Layer2	Copper	0.0152 mm	1.1mm (with copper core)
Core	Core	1.065 mm	
Inner Layer3	Copper	0.0152 mm	
Prepreg	7628*1	0.2104 mm	
Bottom Layer4	Copper	0.035 mm	

0.2mm (7.87 mil) is nominal thickness of 7628 prepreg. Use 7.1 mil as the thickness when the controlled impedance tracks are on top/bottom, use 8.1 mil when tracks are inside.

Fig. 3. The Layer Stackup used for the Brainboard[1]

The top layer contains a large number of the components of the Brainboard. The top and bottom layers are filled with a ground plane for signal isolation. Ground planes are used in PCBs to provide a low−impedance return path for current, improving the electrical performance of the circuit. The second layer is a ground layer providing easy routing through the board for short−distance ground connections. The next layer is a 3.3V power plane. Similar to the ground plane, having a power plane is useful for easy routing, and lower circuit impedance. The bottom layer has decoupling capacitors and traces that could not be routed on the top plane. The three ground planes are ”stitched together” using vias. Vias are used in PCBs to connect signals between layers of the board. In the case of this project, the ground vias are used to not only improve

signal integrity but also for signal isolation. Many are placed in groups around sensitive areas of the PCB such as the crystal and USB lines.

3.1.3 USB Lines

The USB data lines are two of the most important traces on the Brainboard. Because of USB's specific impedance requirements (reference), the traces must be configured as a differential pair. Differential pair traces are traces that are the same width and placed a constant distance apart to maintain the same signal impedance throughout the entire trace. The signal impedance for USB is defined in the USB specification as 90Ω in order to match the USB cable differential impedance as mentioned on page 10 of [6]. JLCPCB provides an impedance calculator which was used to calculate the appropriate trace widths and space for the USB differential pair. The JLCPCB impedance calculator uses data provided for their specific stack−up configurations. Using this calculator, and choosing a trace space of 8 mils, the recommended trace width for the USB lines is 10.28 mils. Figure 4 shows the calculator from JLCPCB's website.

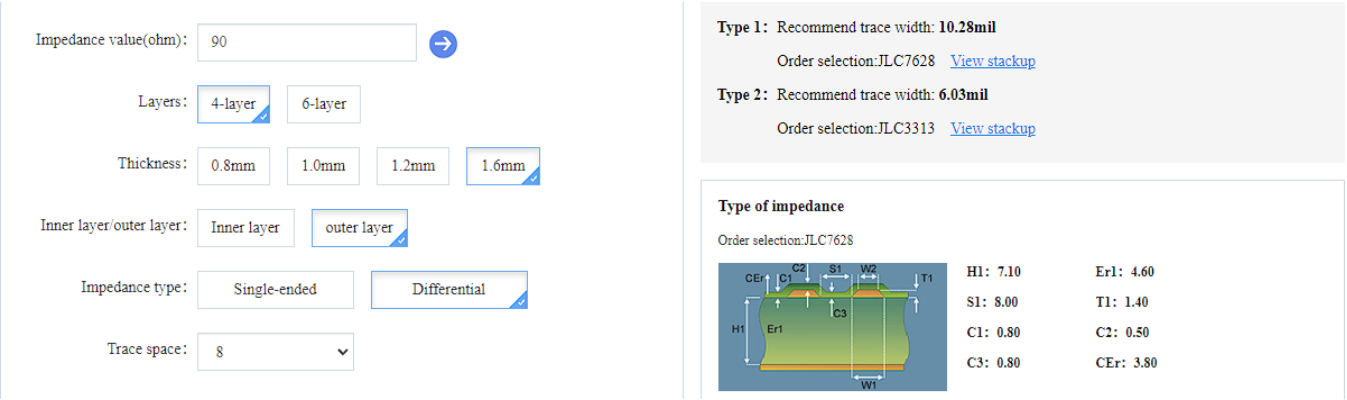

Fig. 4. The differential pair calculator provided by JLCPCB[2]

The USB specification requires 27Ω series termination resistors on the USB data lines.

3.1.4 Microcontroller

This project does not have many technical constraints on the microcontroller used. The primary constraints are the ability to communicate over USB and SPI, as well as at least 6 GPIO pins for the buttons and volume dial. Most available microcontrollers meet these specifications, so instead, a microcontroller was chosen that would be easy to work with and had great documentation. The microcontroller chosen is the RP2040 from the Raspberry Pi Foundation. As mentioned, the RP2040 was not chosen for technical reasons, it does not do anything that most microcontrollers

cannot. However, it has excellent documentation (for both hardware and software), a flexible build system, and it was available at a low cost and in a low order quantity with many in stock. The RP2040 is a QFN−56 package, meaning it is difficult to hand−solder. Because of this, the Brainboard is assembled by its manufacturer JLCPCB.

3.1.5 Decoupling Capacitors

The RP2040 requires decoupling capacitors on each power pin of the chip [6]. The decoupling capacitors are used to filter power supply noise. However, they also serve a purpose in handling sudden current changes. Because they must be placed so close to the pins, the optimal place to put them is directly underneath each pin, on the opposite side of the PCB. Designing a board with parts on both sides increases costs, but placing the capacitors underneath the microcontroller pins allows easier routing and placement of other devices such as the flash memory.

3.1.6 Flash Memory

The RP2040 also requires an external flash to be connected via Quad−SPI. The RP2040 can support at most an external flash chip with 16MB of memory. Additionally, the RP2040 has a Synchronous Serial Interface (SSI) controller which requires the external flash device to have one of the following interfaces. Either Motorola SPI, Texas Instruments SSP, or National Semiconductor Microwire. The W25Q128JVSIM (W25) from Windbond Electronics was chosen because it features the full 16MB available to be used, and communicates over SPI. Additionally, it is also used and recommended by the RP2040 hardware design guide. Other smaller and less expensive Quad−SPI flash memory chips could have been used, however, the W25 was available for use, and is the recommended chip. In order to maintain signal integrity the memory must be placed physically close to the microcontroller. The flash memory is also used to control when the device is placed into boot mode and can be programmed with new firmware. To do this, the microcontroller reads the value of the Quad−SPI$_{SS}$ line. Figure 5 shows a schematic of the flash memory circuit.

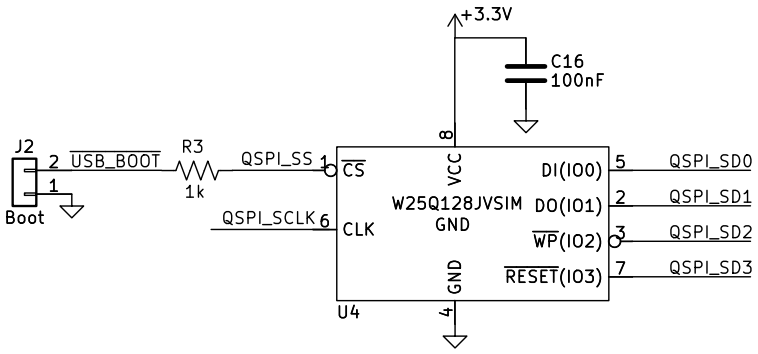

Fig. 5. The circuit schematic for the Flash Memory on the Brainbaord

If SS is read as low, the device enters boot mode, and new firmware can be uploaded. Otherwise, SS reads as high, and the RP2040 executes the code stored in the flash memory. This can be controlled by the user using the BOOT header (J1). Remove the jumper to run flashed code, and place it to reprogram. The RESET button is wired to the RUN pin of the RP2040 to allow the user to reset the device, useful for entering boot mode.

3.1.7 Crystal Oscillator

The RP2040 requires an external 12MHz crystal oscillator. The microcontroller has an internal clock, but the exact frequency can not be guaranteed due to manufacturing differences. A Crystal oscillator may be connected between XIN and XOUT on the RP2040 to provide a more accurate clock. The RP2040 hardware design guide recommends the ABLS−12 which was available for purchase and use, however, the crystal footprint is larger than desired and has a stability of $\pm 30ppm$. The stability determines how much the output frequency will vary with external conditions such as temperature, voltage, and load variations. For this project, the crystal used is the TAXM12M4RLBDDT2T (TAXM12). It was chosen for its better stability of $\pm 20ppm$ and smaller SMD footprint. Crystal oscillators have strict tolerances on load capacitance and ESR. This crystal must have a load capacitance of 20pF with an equivalent resistance of 50Ω. Figure 6 shows the crystal oscillator and its required components.

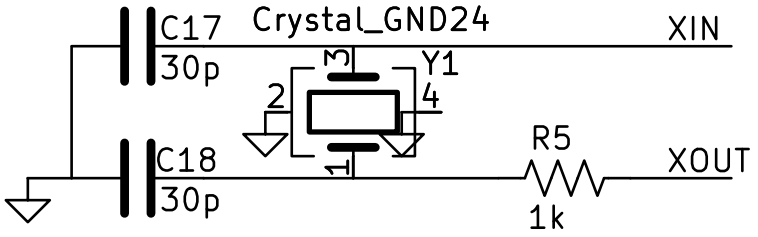

Fig. 6. The circuit schematic for the Crystal on the Brainbaord

Capacitors C_{X1} and C_{X2} are connected in parallel, so the load capacitance of the crystal can be found in equation 1.

$$C_L = \frac{C_{X1} * C_{X2}}{C_{X1} + C_{X2}} + C_S \tag{1}$$

C_S is the system stray capacitance which is the capacitance between the two traces on either side of the crystal. For a standard PCB like the one used for this project, it is common to use 5pF as an estimate. Stray capacitance can be reduced by placing the crystal as close as possible to the XIN and XOUT pins. Therefore using two 30pF capacitors, the load capacitance can be calculated to be equal to 20pF, as required. The 50Ω ESR requirement is a maximum and can be met by using multi−layer ceramic capacitors (MLCC's), which provide low ESR. In addition, a 1k resistor is placed in series with the crystal to limit the drive level of − or the current through it. As passive components, crystals cannot handle high currents without overheating and getting damaged.

3.1.8 Power Supply

5V USB power must be stepped down to 3.3V for use by the device. This can be achieved in numerous ways. To obtain an efficiency of greater than 75% a DC−DC converter is used as this is a well−established design that can get upwards of 90% efficiency. The entire project uses less than 200mA. An off−the−shelf DC−DC converter was used as it lowers the overall parts used and abstracts away many of the complexities that come with designing a custom DC−DC converter. The MCP1603 is a DC−DC converter from Microchip which has a fixed output voltage of 3.3V and an input voltage range of 2.7V to 5.5V. the MCP1603 has a max output current of 500mA, and is specified to operate with an efficiency over 90% when the load draws over 20mA. Figure 7 shows the MCP1603 power efficiency graph as a function of the output current.

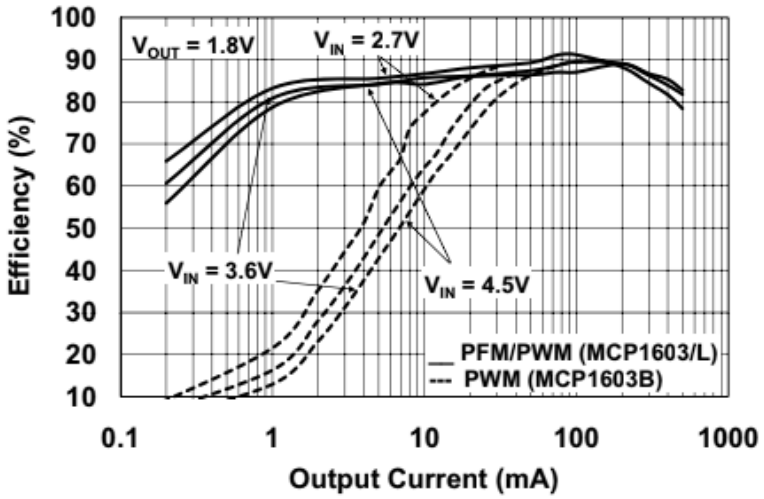

Fig. 7. The power efficiency graph provided on page 2 of the MCP1603 datasheet[3]

The particular MCP1603 used in this project is the PWM−only variant. As can be seen, this requires more current to be drawn from the power supply for higher efficiency. Three external components are required for the MCP1603 to operate correctly, a bypass capacitor for the input and output, and an inductor on the output. The MCP1603 datasheet recommends the input and output capacitance be $4.7\mu F$, and the inductor to be $4.7\mu H$. Figure 8 shows the power supply schematic.

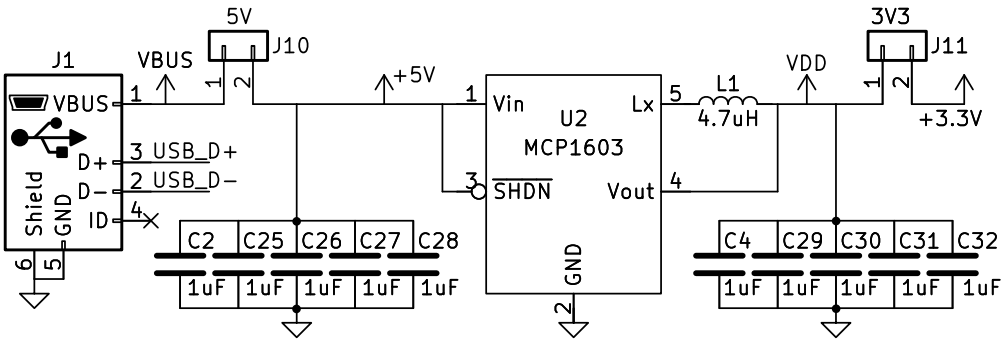

Fig. 8. The circuit schematic for the Power Circuit on the Brainbaord

The MCP1603 switches at a frequency of 2MHz. Due to the self−resonance frequency (SRF) of a capacitor, with an increasing frequency, the effective capacitance drops. This is because as the frequency increases so does the reactance of the capacitor, and once the reactance is equal to its impedance the capacitor acts as an open circuit. Capacitors are specially chosen with a high SRF such that they still operate at their desired capacitance while at the required frequency of 2MHz. To optimize the trade−off between capacitance and high SRF, multiple smaller capacitors can be used for their improved frequency response. In order to meet the required capacitance value of $4.7\mu F$, 5 $1\mu F$ capacitors are connected in parallel, with an equivalent capacitance of $5\mu F$, which is acceptable for the input and output capacitance. Additionally, smaller capacitors have better performance at higher DC bias conditions than larger capacitors. The capacitors chosen for the power supply are the CL21B105KBFNNN's from Samsung, for their SRF of 10MHz, and strong DC bias characteristics.

The inductor also had some design considerations past the given inductance value. Although the datasheet did not specify it, a shielded inductor with a keep−out layer underneath it was used to improve circuit performance. A shielded inductor is designed in such a way as to reduce the amount of magnetic flux leaked to the surrounding area. This is done to reduce the effects of electromagnetic interference (EMI). Additionally, shielded inductors can often handle more power and are frequently used in power applications. A keep−out layer is a section of the PCB where no copper is allowed to enter. This is used to further reduce the effects of EMI. Similar to the capacitor, an inductor also has an SRF at which it no longer behaves like an inductor. Therefore, an inductor had to be chosen with a high SRF like the capacitors, for the same reasons. The inductor used for this project must have a rated current high enough for all of the other components used. Current draw for the HEAT MIDI Keyboard was estimated as follows. Raspberry Pi gives

an estimated current draw for the RP2040 of 25mA while under load. [7]. Each hall effect sensor has an average current draw of 3mA[8], giving a total of 36mA for all twelve sensors. The buttons and rotary encoder are configured as active low devices, meaning when they are pressed or rotated, the voltage drops to 0V. This means while they are not pressed they are constantly drawing a small current. For the buttons, with a resistance of 2kOhms, at 3.3V, the current is determined to be around 1.6mA. With three buttons this results in a current draw of around 5mA. Similarly, the rotary encoder is estimated to have a current draw of 10mA. The datasheet for the analog−to−digital converter lists the supply current at 1.8mA for the use case of 3.3V supply voltage, and 1MHz throughput.[9]. The display datasheet is incomplete and does not list power consumption values. Instead, the current draw is estimated by measuring the output current with and without the display connected to the brainboard. With this method, the current draw is estimated to be around 15mA. Four LED's are also connected to the RP2040, to increase current draw for better efficiency in the power section. These are limited by 330Ohm resistors meaning each LED will draw around 10mA when powered on. Using these values, the total current draw for the HEAT MIDI Keyboard is estimated to be near 150mA. The inductor used for the power circuit is the LQH43PN4R7M26L from Murata, chosen for its high SRF of 35MHz, and high rated current of 1.4A, much higher than required for this project.

The RP2040 has a required power supply ripple. For general operation, the input voltage range is generous. However, for USB functionality to work properly the USB−PHY pin must be given 3.3V $\pm 5\%$. Equation 2, provided on page 17 of the MCP1603 datasheet[3], can be used to determine the voltage ripple of a buck converter.

$$\Delta Vout = \Delta I_L * ESR + \frac{\Delta I_L}{8 * F_{SW} * C_{out}} \tag{2}$$

Here, ΔI_L is the current ripple through the inductor, ESR is the equivalent series resistance of the output capacitor, F_{SW} is the switching frequency, and C_{out} is the capacitance of the output capacitor. Most of these values are known. The specific Samsung capacitors used have an ESR of 8.5mΩ, as specified in the datasheet. With a switching frequency of 2MHz, and output capacitance is 4.7uF the only value that must be calculated is the inductor current ripple. The MCP1603 datasheet[3] also provides an equation on page 18 to find this:

$$\Delta I_L = \frac{V_{out}}{F_{SW} * L} * (1 - \frac{V_{out}}{V_{in}}) \tag{3}$$

Here, with an input voltage of 5V, and an output voltage of 3.3V, the aforementioned switching frequency of 2MHz, and an inductance (L) of 4.7uH, ΔI_L is 119mA. Finally with all of these values, the voltage ripple ΔV_{out} is calculated to be 2.59mV. This is well within the necessary range for the RP2040 to function properly.

The only criterion for the display is that it is able to tell the user the velocity of the most recent key press. There are many ways to do this, such as a seven−segment led display or a character LCD. It is desirable to be able to display both text and characters, as well as to be able to see multiple lines of text at a time to provide more information during development. With these

considerations, a 1.8 inch, 128 by 160 pixel LCD from Adafruit was chosen. This display was chosen because it is highly available, has good documentation, and is a self−contained unit.

3.1.9 User Input

The final circuits of the Brainboard are the user input buttons, and dial, which must be debounced. Switches bounce when closed, which may cause multiple button presses to be registered when the user only intended to press the button once. This is no different for a rotary encoder, the device used for the volume dial. To avoid this, both circuits must be debounced, either in software or hardware. For this project, hardware debouncing is used to ensure the software on the RP2040 only sees clean signals, simplifying the software development process. Hardware debouncing is performed with an RC circuit. Figure 9 shows a debounce circuit attached to one of the user input buttons.

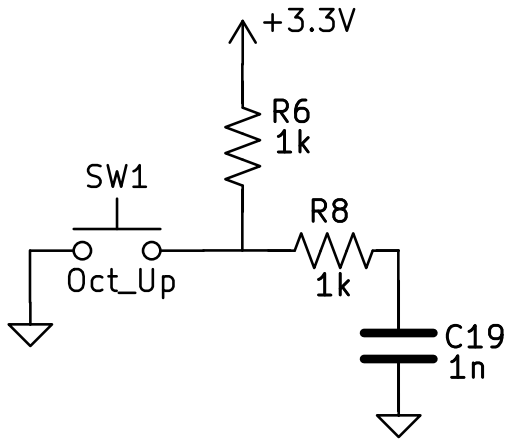

Fig. 9. The circuit schematic for the button debounce circuit on the Brainbaord

Figure 15 shows the rotary encoder schematic.

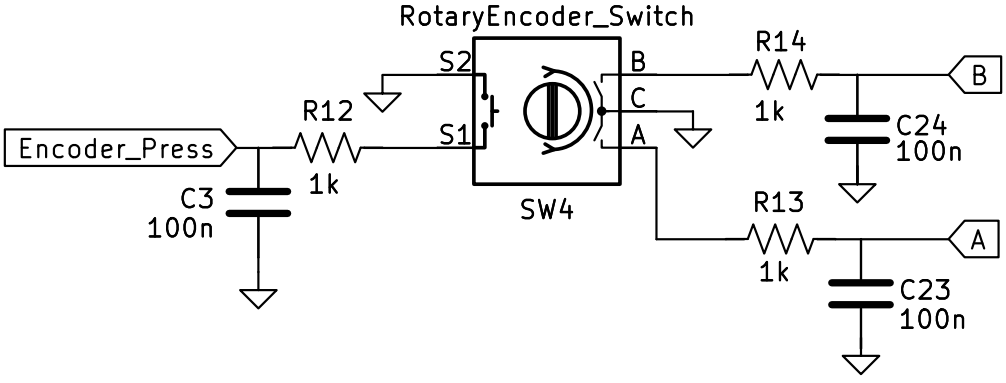

Fig. 10. The circuit schematic for the rotary encoder debounce circuit on the Brainbaord

The rise and fall time of the circuit can be found with Equation 4.

$$\tau = R * C \tag{4}$$

τ is the time constant of an RC network like the one used for the debounce circuits. It can be used to find the total time for the signal to rise or fall. Equation 5 shows this calculation.

$$t = 5 * \tau \tag{5}$$

Five τ is defined as "a long time" and can be used as the rise and fall time for the debounce circuits used. The button circuit has two resistors, $R6$, and $R8$, and a capacitor $C19$. Both resistors have a value of 1kΩ, and the capacitor has a value of 1nF. When the switch is open i.e. the button is not pressed, the capacitor charges through both resistors. This results in a time constant τ of 2μs, which is a rise time of 10μs. When the switch is closed, the capacitor only discharges through one resistor. This results in a time constant τ of 1μs, and a fall time of 5μs. These are both acceptable values for a simple debounce circuit. The rotary encoder is similar, however it only has one resistor, so the time constant τ is the same for the open and closed state. The rotary encoder also uses a larger capacitor, and the same resistor value. This results in a longer time constant τ of 100μs, and a rise/fall time of 500μs.

Other considerations on the Brainboard include the board bulk capacitor, which is a 10uF capacitor connected between power and ground to help reduce ground noise. Additionally, four LEDs are connected to current limiting 330Ω resistors. These LEDs are used as indicators for the user. Standard 0.1in spaced pin connectors are used for connecting the boards together, as well as connecting the display to the Brainboard.

3.1.10 Keysboard

The Keysboard is much simpler relative to the Brainboard. It has twelve hall effect sensors which output an analog signal relative to the strength of the magnetic field near them. The Keysboard has twelve 3D printed keys, twelve hall effect sensors, an analog−to−digital converter (ADC), and a reference circuit.

3.1.11 Analog to Digital Converter

Each key has a small magnet positioned such that when a key is pressed the magnet moves closer to the hall effect sensor thereby changing its analog output voltage. These analog signals are passed to the ADC. The ADC must have at least 12 channels − one for each hall effect sensor, and it must be able to sample fast enough to detect the fastest key press. There is no need for a high−resolution high−precision ADC as the only thing being measured is the fall time of the key. The ADC selected is the ADS7960, an 8−bit, 12−channel ADC from Texas Instruments. The

ADS7960 communicates via SPI, and can sample at 1MHz, plenty fast enough for a key press. The ADS7960 requires an external reference voltage of 2.5V. the LM4040DBZ−2.5 is a fixed output voltage reference that only requires one current limiting resistor. The LM4040 operates between 1mA and 15mA, the resistor is calculated in Equation 6:

$$R_s = \frac{V_s - V_r}{I_l + I_q} \tag{6}$$

I_q is the maximum current of the LM4040, I_l is calculated from the input resistance of the ADC which is $100k\Omega$, with a voltage drop of 2.5V which equals a current of $25\mu A$. V_s is the supply voltage or 3.3V and V_r is the regulated voltage of the LM4040 which is 2.5V so the smallest resistor that can be used is 53Ω. Using the same equation but with I_q being 1mA the largest resistor possible is 780Ω. Therefore, a 237Ω resistor is used.

3.2 Software

The MIDI Keyboard uses both cores of the RP2040. When a key is pressed Core 0 must make Core 1 aware of this change. The Raspberry Pi Foundation offers a thread−safe queue in the SDK[10]. This Queue is set up so that Core 0 is the only core that queues data onto it and Core 1 is the only core that dequeues data from it. A queue is used instead of an atomic variable to avoid the possibility that Core 1 has not handled the previous keypress when Core 0 is ready to push it. This also allows Core 0 to hand off its data and immediately begin its next tasks. The RP2040 SDK provides many libraries to make use of the hardware such as hardware APIs for the GPIO, timers, and SPI as well as TinyUSB, a library for developing USB devices [10].

The main program starts on Core 0. The first tasks are hardware initialization, initializing standard IO, General Purpose IO (GPIO), initializing the ADC, initializing TinyUSB, and then starting Core 1. After the initialization, Core 0 enters a loop where it has three tasks: the TinyUSB task, MIDI task, and keyboard task. The TinyUSB task is a function provided by TinyUSB to handle all communication via the USB port. The MIDI task handles incoming MIDI messages from the host device. The Keyboard task analyzes the current state of the keyboard and determines if a key is being pressed, released, or sustained and generates MIDI messages accordingly. The MIDI task is also where Core 0 queues data for Core 1.

Once Core 1 is started it initializes the display and enters its own loop. This loop waits for data to be available on the queue and calculates the velocity in centimeters per second. With the velocity in centimeters per second, the main loop draws text on the screen to tell the user the velocity of the most recent key press. Along with the velocity, other diagnostic information is drawn on the screen including the voltage at which the key press starts, the ADC value of the detected key press, the time elapsed while the key was falling, and the key that was pressed (0−11).

3.2.1 Display Driver

The display is 128 by 160 pixels. There are multiple color modes for the display ranging from 4000 colors to 262,000 colors. As the color range increases the amount of RAM to store the display buffer must also increase. To conserve memory the minimal color mode was chosen. The display buffer is an array of 8−bit unsigned characters. When using the display in 4K color mode each sub−pixel is allotted 4 bits. There are 4 bits for Red, 4 for green, and 4 for blue. With the display buffer being an array of 8−bit values one and a half bytes are used per pixel. The sub−pixels are interleaved with the following pattern where B is the display buffer, $B[n]$ is the n−th 8−bit value in B and N is the length on B. Equation 7 shows the layout of this buffer.

$$B[n] = \{R_1G_1, B_1R_2, G_2B_2, R_3G_3, ..., R_{N-1}G_{N-1}, B_{N-1}R_N, G_NB_N\} \tag{7}$$

The display buffer size is $128 * 168 * 1.5$ which is 32kB. To make the display easier to use the driver is designed to abstract away the display buffer. Instead, the users of the driver interface with an array that is exactly the length of the number of pixels on the display. Each element is a 16−bit integer where the lower 12 bits represents one pixel, and each subpixel is allotted the last 4 bits. Equation 8 shows the pattern where D is the user facing display, $D[m]$ is the value of a certain pixel and M is the length of D.

$$D[m] = \{R_1G_1B_1, R_2G_2B_2, ..., R_MG_MB_M\} \tag{8}$$

This means there must be a way to convert from D to B, where the 16−bit values are divided into 8−bit values and interleaved properly for the display to accept it. Algorithm 1 shows how this is done.

The display driver is only able to draw two things: characters and rectangles. Each character of the font is a five−pixel by seven−pixel glyph. The font is encoded as a series of 8−bit values where one character is five bytes wide. Each bit is associated with a pixel, if the bit is high the pixel is on. The characters are stored contiguously in an array to align with ASCII character ordering. For example, to find the character 'a' it would be the five bytes from location 325 to 330 because the ASCII value for 'a' is 65, and $65 * 5 = 325$. The font is from Adafruit.

Rectangles are drawn by providing a start location for both the X and Y axis, along with width and height values. One of the many commands offered by the display is to move its origin to a different location on the screen. When drawing a rectangle, the origin is moved to the X and Y set by the user and a row of pixels the width as specified by the user is drawn and the origin is moved down one pixel. This process is repeated until the origins Y value has moved the same amount of pixels as the height.

Writing to the display is done with SPI, but the display has an extra control line to differentiate between commands and data called DC. Commands are for controlling aspects of the display

Algorithm 1 Conversion from driver array to display buffer.

$acc \leftarrow 0$
$i \leftarrow 0$
$bi \leftarrow 0$
while $i < M$ do
    if acc is empty then
        $B[bi] \leftarrow (D[i] >> 4)\&hFF$
        $acc = D[i]\&hF$
        $bi \leftarrow bi + 1$
        $i \leftarrow i + 1$
    end if
    if top half of acc is set then
        $B[bi] \leftarrow acc$
        clear acc
        $bi \leftarrow bi + 1$
    end if
    if Bottom half of acc is set then
        $B[bi] \leftarrow ((acc << 4)\&hF0)|((D[i] >> 8)\&h0F)$
        $acc \leftarrow D[i]\&hFF$
    end if
end while

such as orientation, software resets, color modes, etc. A command can take a range of parameters which are sent as data over the SPI bus. Typically a command takes anywhere from zero to four bytes of parameters with the exception of the command for writing pixels, which takes at most the size of the display buffer. When the user requests to write to the display, a command, a pointer to data, and the length of the data must be passed. The DC line is pulled low to write the command, it is then pulled high to write the data.

3.2.2 Analog to Digital Converter Driver

Like the display, the ADC communicates via 16−bit SPI. To avoid collisions with the display the ADC has been separated and put on the second SPI bus. The ADC has three modes of operation: manual, auto 1, and auto 2. Manual mode is used when the channel being sampled is arbitrary and is selected by writing the channel to read within the 16−bit SPI packet. Auto−1 is used to sample the same non−contiguous channels on the ADC which is set while programming the ADC. Auto−2 is used to sample 1 or more contiguous channels starting from channel zero up to the channel specified while programming. This project uses auto−2.

To send and receive data from the ADC timers and interrupts are used. A repeating timer sends a command to the ADC to report the next channel in the sequence, this is done by writing the

command to the SPI write register. An interrupt request is set up to read the response from the ADC when it is ready. The timer to write to the ADC triggers every $15\mu s$, this is a fast enough time that the ADC is able to loop through all keys in $180\mu s$ and determine an accurate fall time which directly affects the accuracy of the velocity in cm/s. The interrupt routine is also responsible for determining if a key has been either pressed or released and updating the start and stop times of the current key being pressed, which is used for determining velocity (both for MIDI and in cm/s). A key press is marked as starting if the ADC reports that channel below a certain value − in this case 600mV − and if the current voltage is smaller than the previous voltage, which is done to determine if a key is falling. The end timestamp is determined when the ADC reads a value lower than 30mV.

3.2.3 Velocity Detection

As mentioned in Section 3.1, the movement of a key is detected by reading the analog values of a hall effect sensor in line with a magnet that is attached to the key being pressed. These magnets have varying field strengths. So although the keys are all triggered at 600mV, they do not trigger at the same height. The trigger height for each key was measured with digital calipers. The trigger height measurement was done by placing the calipers between the key and the PCB. The voltage of the hall effect sensor was measured on an oscilloscope, as the key moved down the hall effect sensor voltage dropped and the distance the caliper reported reduced. The key was pressed until the oscilloscope read 600mV, and the measurement of the calipers was recorded. This process was repeated for all keys. A look up table was stored in memory on the RP2040 to map a key to the height that it triggers at. The lookup table stores the trigger heights in centimeters as a floating point value. Using the lookup table and the fall time for a key, the velocity in centimeters per second can be calculated. Once the velocity is calculated the RP2040 writes on the display.

3.2.4 Generating MIDI packets Over USB

Much of the MIDI code is abstracted away by the TinyUSB library, however it is still important to generate an appropriate MIDI packet. MIDI packets are sent as up to 3 byte long sequences. The first byte is the status byte which determines what the function of the packet is. The following bytes are the data bytes which contain the information needed for the specified function. For any note to be played and stopped on the host device: First a Note On (ON), then a corresponding Note Off (OFF) message must be sent. In a standard ON message, there is the note on status byte, and two data bytes. The first data byte represents the pitch value, or more simply, the note, and the second byte represents the velocity. The OFF message is unique, as either the specific note off status byte can be used, or more simply, an ON message with a velocity of 0. Both methods have the same effect in the MIDI specification. MIDI also supports other types of messages other than notes. These are called MIDI controllers, and they can adjust a number of parameters on the host device such as volume, pitch modulation, and panoramic sound (left and right channel).

For this project, only two MIDI controllers were used, volume control, and mute. To send a volume change packet, the first byte is again a status byte. The next byte determines which of the pre−defined 128 controllers will be used. Note that despite 128 being available, very few are used in practice. For volume control, controller number 7 is used. The last byte contains the value to be assigned to the controller, again an 8 bit value ranging from 0 to 127. The last 8 controllers have a special name, channel mode messages, and are used for very specific functions on the given MIDI channel. This project uses the ”All Sound Off” channel mode message to drive a mute button for the device. This effectively sends a note off for all notes on the current channel, and sets their volumes to 0, cutting the sound, instead of letting it fade out.

3.2.5 Reading Other User Input

A number of other user inputs are available to the user. Octave and volume must also be con−trolled. Octave is adjusted locally on the keyboard itself, while volume must be sent as a MIDI message to the host device. Buttons are used for controlling the octave, and a rotary encoder is used for the volume. These are all connected to GPIO pins on the RP2040. In order to respond quickly to user input, GPIO interrupts were used. The SDK provides a high level interface to interact with GPIO pins and setup their corresponding interrupts. In the RP2040, the GPIO pins share a common interrupt handler, meaning they must use the same callback function. The call−back function is a function called whenever the specified pin has a GPIO ”event”. GPIO events are what the RP2040 calls transitions of the GPIO signal such as high, low, rising, and falling. The callback function receives the event that triggered the interrupt, as well as the number of the GPIO pin that had the event. A switch statement is used to perform different tasks based on the GPIO number that called the interrupt. This means that all code within the switch statements must occur quickly as to make sure the next interrupt can be captured in time.

4 Results

This section will detail the results of testing done on the project. Testing was done to ensure the project met contract specifications as detailed in the introduction. These specifications can be divided into four tests: Power Efficiency, PCB, Velocity Accuracy and, MIDI Compliance. The following is a photo of the final product, with the brain board connected directly to the keysboard, and both keys and display attached.

4.1 Power Efficiency

The first specification to test is the power efficiency of the power supply. The overall power in and out must be determined to calculate efficiency. Therefore, the voltage and current at both the input and output must be measured. This is done with 4 multi−meters connected to the input and output of the power regulation section on the device. These values can be used to determine the power in and out of the device with Equation 9

$$P = V * I \tag{9}$$

Here power is P, voltage is V, and current is I. The voltage across the input is $5.07V$, and the voltage across the output is $3.10V$. The input and output currents are 50.8mA and 69.5mA, respectively. This results in a power in of 257.56mW, and a power out of 215.45mW. These values are used in Equation 10 to find the total power efficiency.

$$Efficiency = (\frac{P_{out}}{P_{in}}) * 100 \tag{10}$$

Here P_{out} and P_{in} are the power in and power out calculated above. These values result in a total power efficiency of 86.1%. This is well above the required 75%, therefore the specification is met. It is important to note potential error with this test. Due to the multi−meters used there is the potential for measurement errors in the device, as they are only accurate up to 4 digits. The device was measured during normal operation, however if more components on the device were active such as the LED's and sensors, then the power draw could have increased and effected efficiency. However, it should also be noted that the efficiency curve for the DC−DC converter is shown by Figure 11.

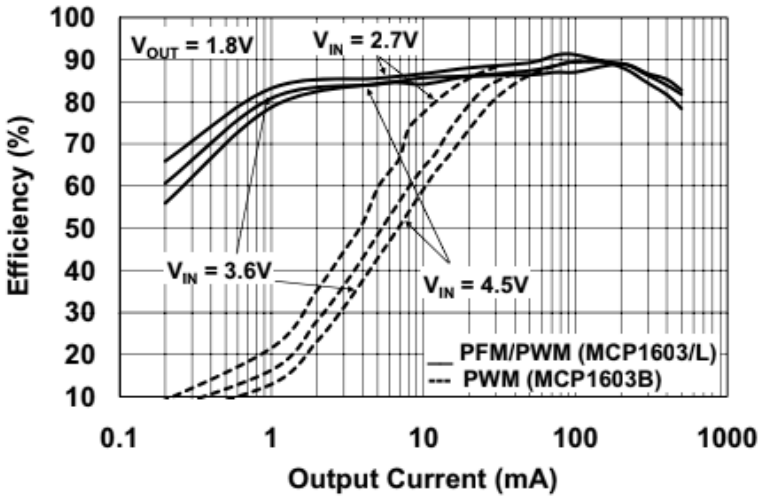

Fig. 11. The power efficiency graph provided on page 2 of the MCP1603 datasheet[3]

As can be seen, drawing more power from the device could potentially improve efficiency, rather than hinder it.

4.2 PCB

The next test covers two of the contract specifications simultaneously. The specifications are that the device must make use of a custom PCB and not use a pre−manufactured development board. As can be seen from the following figure, the device is on custom built PCB with no pre−manufactured development boards.

The PCB's were designed as separate boards to improve compatibility with potential future attachments.

4.3 Velocity Accuracy

This subsection will describe the testing done to ensure the displayed velocity is accurate to the specified 5−20cm/s±10%. Because the height of each key is different, the fall time is measured to determine accuracy of the keyboard sensors. To measure the fall time, an oscilloscope probe is attached to the analog output of a given channel on the keyboard. When the key is pressed, the voltage falls, reaching near zero at the bottom of the key press. The fall time is determined from the oscilloscope trace by measuring the time between the trigger level voltage, and the bottom voltage. Figure 12 shows an example of such an oscilloscope trace, with the appropriate levels labeled.

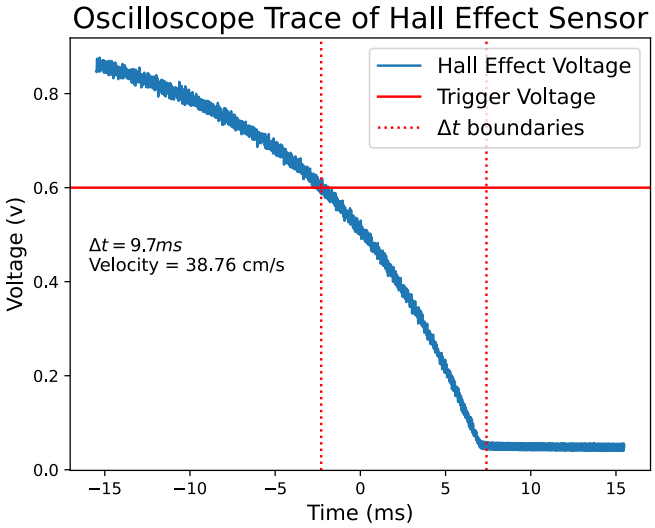

Fig. 12. The oscilloscope trace of a hall effect sensor detecting a keypress.

The trigger level voltage is calculated by the microprocessor and also printed to the display to aid in testing. Using the measured fall time on the oscilloscope, the velocity can be calculated using

Equation 11.

$$Velocity = \frac{FallDistance}{FallTime} \tag{11}$$

The velocity is calculated to be 38.76 cm/s. The device reported a velocity of 36.24 cm/s for this key press, with an error of 6.95%, meaning the specification is met. See Figure 13 for a photo of the velocity displayed on the LCD.

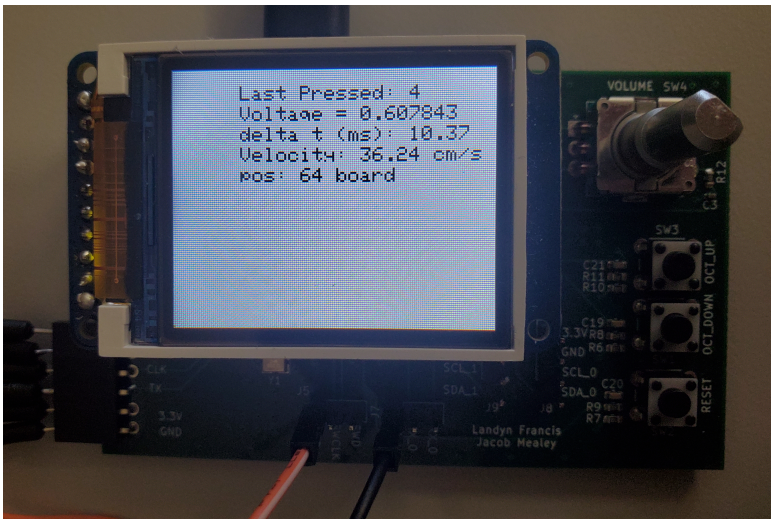

Fig. 13. The velocity as displayed on the screen.

4.4 MIDI Compliance

This section describes how the MIDI protocol compliance was tested for the MIDI Keyboard. As mentioned in the introduction, MIDI protocol compliance for this project is defined as successfully sending MIDI signals from the keyboard to host device, resulting in playback of the desired note. Following the setup steps detailed in Section 1, the device can be connected to a DAW program for playback. During testing, the device successfully connected to the DAW program and sends MIDI messages. The MIDI compliance can also be tested using a standalone MIDI reader software. On Linux, the ‘aseqdump‘ command can be used on port 20 to print out incoming MIDI messages to the terminal. On Windows, MIDIView is a program that can be installed that also prints incoming MIDI messages. The device shows up in all three programs, and therefore meets the MIDI compliance specification.

5 Conclusion

All of the contract specifications for the Hall Effect MIDI Keyboard have been met.

The keyboard takes advantage of hall effect sensors and small magnets to allow the user to send MIDI messages over a USB connection to a host device running a DAW program. The user is able to play multiple keys at once with minimal sound delay, as well as control the volume of the host device. The power section exceeds the required 75% efficiency, attaining a level of 86.1%. The keys are able to detect key press velocities in a higher range than the original 5−25cm/s specified, as shown by it detecting a speed of 36cm/s . Lastly, the keyboard was designed across multiple custom designed PCB's to allow for future compatibility with other future attachments.

Given more time, increased functionality could be added to the original Hall Effect Always Tracking MIDI Keyboard (HEAT MIDI Keyboard). For example, the mute button acts as more of a volume kill switch, unlike a mute button on a TV, which toggles the volume to 0, then back to the previous volume with a second press. Additionally, MIDI supports pressure sensitivity for keys, allowing vibrato effects to be added for notes played. The hall effect approach on the HEAT MIDI Keyboard actually lends itself well to this purpose, as the brainboard always knows the position of the key. Therefore, with more software, pressure messages could be supported relatively easily. Looking further, the Brainboard itself is designed as a development board for the RP2040 designed by us. Standing alone, it has a wide variety of options in terms of flexibility. Other SPI based instrument devices could be designed and attached to the Brainboard, such as a small and portable MIDI drum kit.

Appendices

Appendix A Contract

Description

A microprocessor based device that meets the MIDI (Musical Instrument Device Interface) protocol will be designed, built and tested in this project. The device will have a full octave (twelve keys) of inputs, which will be sampled and processed in the microcontroller and transferred to a MIDI host device. Along with the twelve velocity sensitive keys, the devices will also have two inputs for increasing and decreasing the octave, with an additional volume dial. The current key velocity will be shown on a display

Inputs

- 5V power via USB
- 12 velocity sensitive piano keys played by user
- Octave Selected by user
- Volume Dialed in by user

Outputs

- Display the current key velocity.
- MIDI data encapsulated over USB.

Specifications

- Velocity detection of 5−20cm/s±10%
- Standard MIDI protocol compliance.
- No development board.
- Custom PCB
- 75% or better efficiency for the power regulation section

Appendix B Bill of Materials

TABLE I
Bill of materials for HEAT MIDI Keysboard.

Item	Part No.	Description	Qty.	Unit Price ($)	LCSC
1	0603B105K500NT	50V 1uF X7R ±10% 0603 MLCC	12	0.17	C90540
2	CL10A106MA8NRNC	25V 10uF X5R ±20% 0603 MLCC	1	0.26	C96446
3	PZ254R−11−O6P	Shrouded Square Pins 2.5mm 6mm 1x6P	1	0.66	C492414
4	KH−A2541WV−12P	250V 3A Straight Square Pins 2.54mm 6mm 1x12P	1	0.2636	C2833334
5	CR1210F237RP05Z	500mW Thick Film Resistor ±1% 237OHM 1210	1	0.25	C174455
6	ADS7960SDBT	TSSOP−38−4.4mm Analog to Digital Converter	1	5.56	C1543353
7	LM4040CIM3−2.5	2.5V ±0.5% 15mA Fixed SOT−23 Voltage Reference	1	1.77	C134019
8	DRV5053VAEDBZTQ1	SOT−23 Hall Sensor	12	1.62	C2866569

TABLE II
Bill of materials for HEAT MIDI Brainboard.

Item	Part No.	Description	Qty.	Unit Price ($)	LCSC
1	CL05B104KB54PNC	50V 100nF X7R ±10% 0402 MLCC	16	0.0067	C307331
2	CL21B105KBFNNNE	50V 1uF X7R ±10% 0805 MLCC	11	0.0079	C28323
3	0603CG300J500NT	50V 30pF C0G ±5% 0603 MLCC	6	0.0038	C1658
4	NCD0603G2	Emerald 0603 LED	4	0.0242	C87326
5	10118194−0001LF	USB 2.0 1 5 Female Micro−B USB Connector	1	0.2190	C132563
6	KH−2.54PH180−1X2P	3A Straight Square Pins 2.5mm 6mm 1X2P	3	0.0172	C2905434
7	PZ254R−11−06P	Shrouded Square Pins 2.5mm 6mm 1x6P	1	0.0335	C492414
8	KH−2.54PH180−1X10P	3A Straight Square Pins 2.5mm 6mm 1X10P	1	0.0891	C2905488
9	KH−2.54PH180−1X3P	3A Straight Square Pins 2.5mm 6mm 1X3P	2	0.0179	C2932698
10	KH−2.54PH180−1X4P	3A Straight Square Pins 2.5mm 6mm 1X4P	2	0.0358	C2905435
11	LQH43PN4R7M26L	1.4A 4.7uF ±20% 75mOHM 1812 Inductor	1	0.2469	C701275
12	0402WGF270JTCE	62.5mW Thick Film Resistor ±1% 27OHM 0402	2	0.0005	C25100
13	0603WAF1001T5E	100mW Thick Film Resistor ±1% 1kOHM	16	0.0005	C21190
14	1825910−6	50mA Straight SPST 24V Tactile Switch	3	0.0669	C428628
15	ECE11E1534408	Plugin Rotary Encoder	1	1.8137	C278348
16	MCP1603T−330I	Step−down Type TSOT−23−5 DC−DC Converter	1	2.3959	C150803
17	W25Q128JVSIM	SOIC−8−208 NOR FLASH	1	1.9187	C2613930
18	RP2040	264KB 30 ARM Cortex−M0 133MHz LQFN−56(7x7) Microcontroller	1	1.1773	C2040
19	TAXM12M4RLBDDT2T	12MHz 20pF ±20ppm SMD3225−4P Crystal	1	0.0628	C133334

Appendix C Schematics

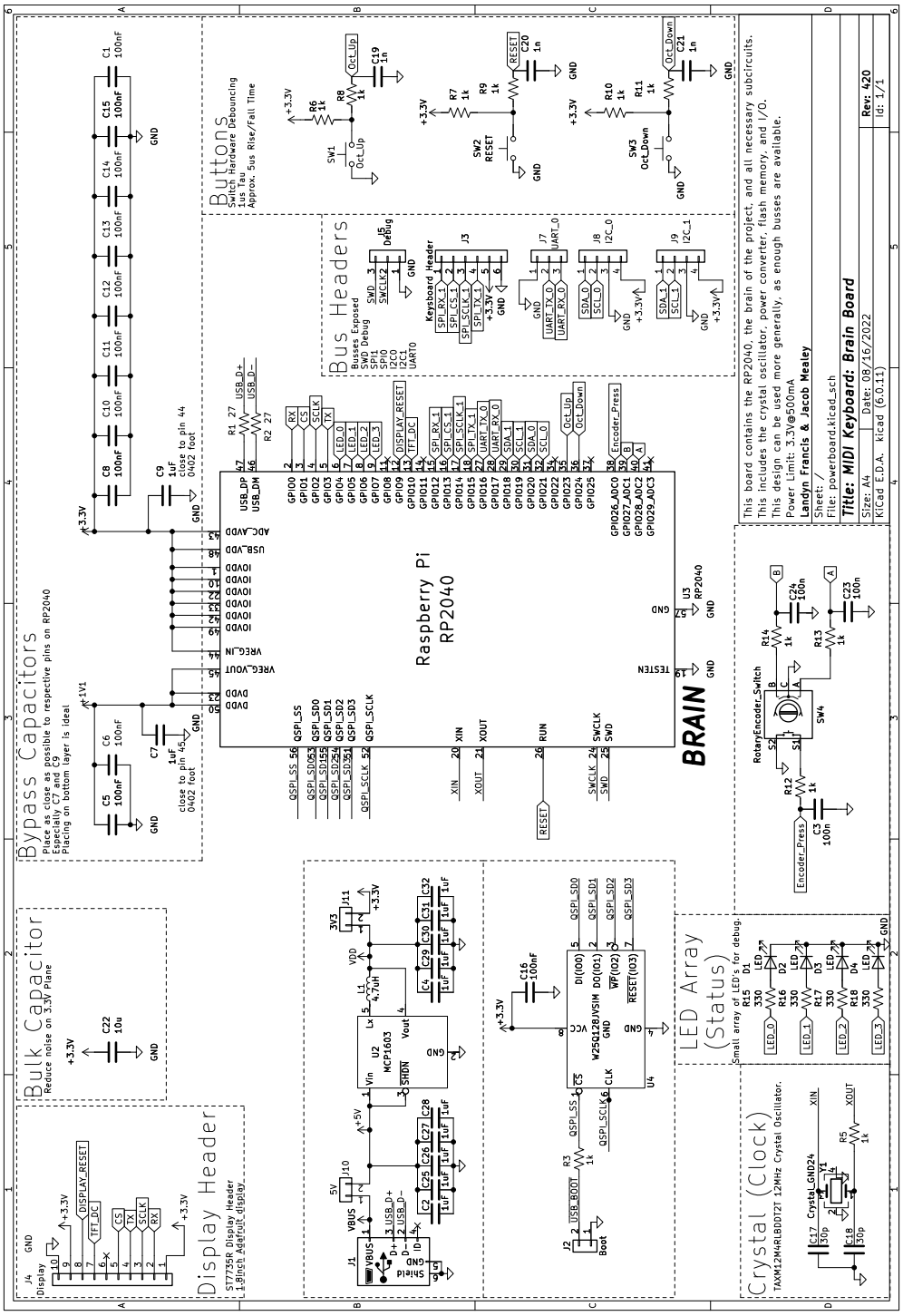

Fig. 14. The complete schematic of the brainboard

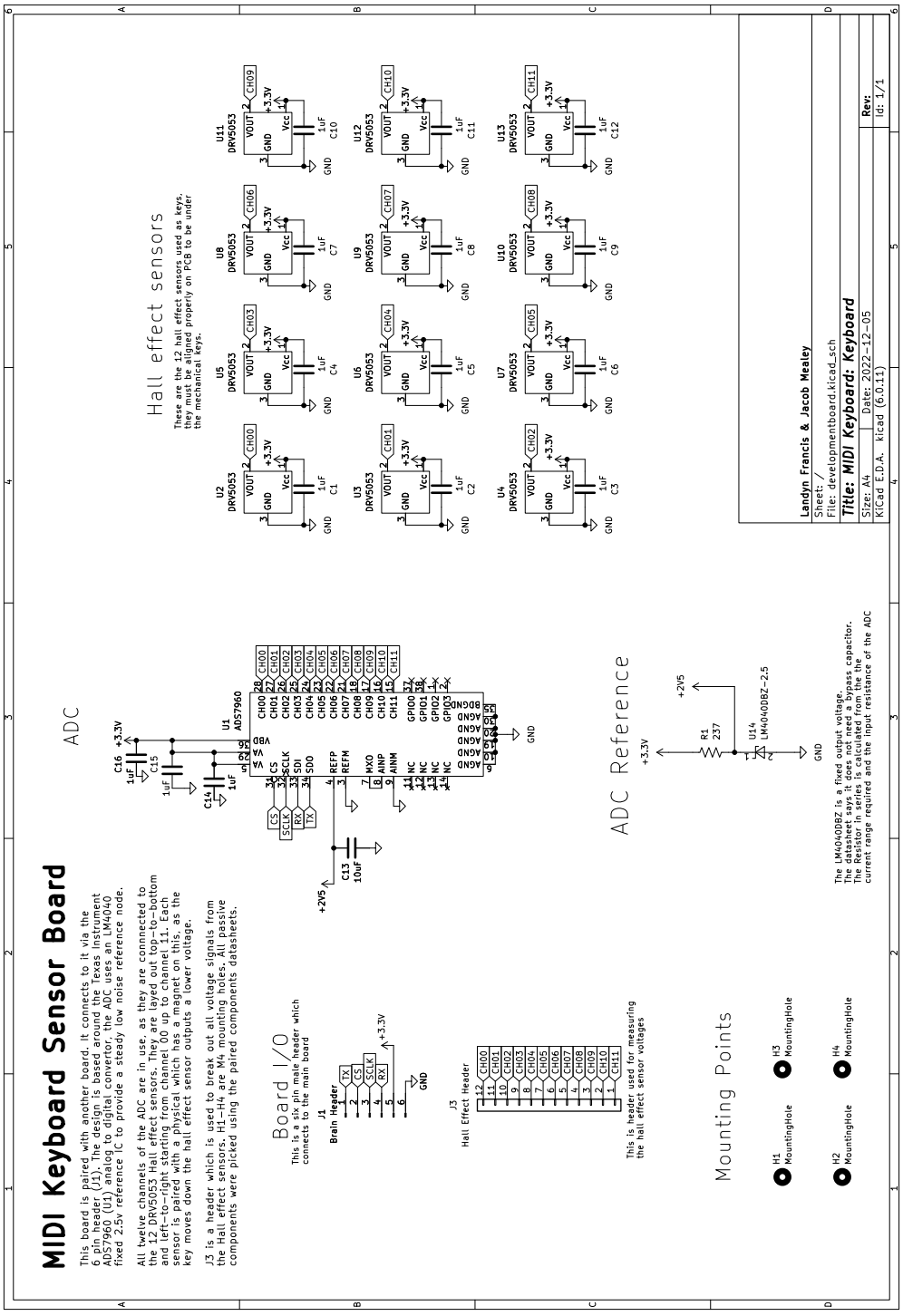

Fig. 15. The complete schematic of the keysboard

References

[1] JLCPCB, “Multilayer high precision pcb’s with impedance control,” 2023. [Online]. Available: https://jlcpcb.com/impedance

[2] ——, “Impedance calculation,” 2023. [Online]. Available: https://cart.jlcpcb.com/impedanceCalculation

[3] Microchip, “Mcp1603/b/l datasheet,” 2012. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/22042B.pdf

[4] T. M. M. Association, “The complete midi 1.0 detailed specification,” 1996. [Online]. Available: https://www.midi.org/specifications-old/item/the-midi-1-0-specification

[5] G. K. Mike Kent, “Universal serial bus device class definition for midi devices,” 1999. [Online]. Available: https://www.usb.org/sites/default/files/midi10.pdf

[6] R. P. Foundation, “Hardware design with rp2040 using rp2040 microcontrollers to build boards and products,” 2023. [Online]. Available: https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf

[7] ——, “Rp2040 datasheet a microcontroller by raspberry pi,” 2023. [Online]. Available: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf

[8] T. Instruments, “Drv5053 analog-bipolar hall effect sensor,” 2014. [Online]. Available: https://www.ti.com/lit/ds/symlink/drv5053.pdf

[9] ——, “Ads79xx pin compatible, 12-, 10, 8-bit, 1-msps, 16-, 12-, 8-, 4-channel, single-ended, serial interface adcs,” 2018. [Online]. Available: https://www.ti.com/lit/ds/symlink/ads7960.pdf

[10] R. P. Foundation, “Raspberry pi pico c/c++ sdk libraries and tools for c/c++ development on rp2040 microcontrollers,” 2023. [Online]. Available: https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-c-sdk.pdf